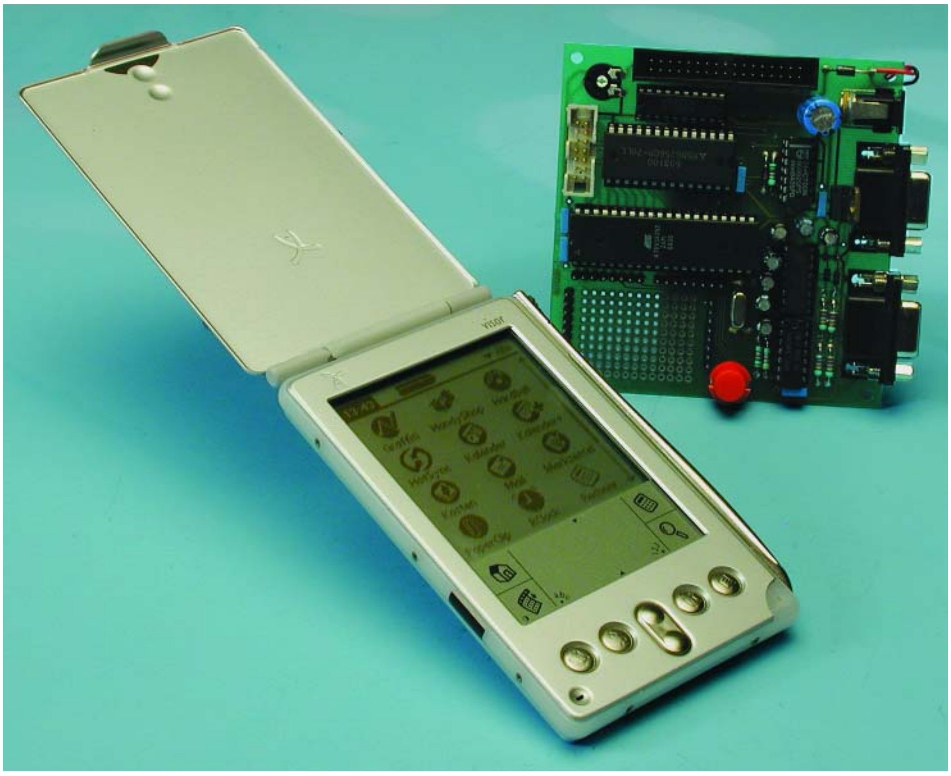
IrDA Interface

Wireless communication from PCB to PDA

By B. Kainka

Wireless communication using infrared light has many advantages over the traditional mess of cables. This goes not only for PCs and laptops, but even for the humble microcontroller.



In particular, it is easier to set up communications with handheld PDA-style computers using infrared than using the RS232 interface. Some newer machines such as the Palm Zire do not even have an external RS232 interface and instead use a USB port for communications with a PC and an IrDA port to talk to other PDAs, mobile phones and other devices. It therefore seems a good idea to equip microcontroller systems such as the *Elektor Electronics* 89S8252 Flash Microcontroller Board (December 2001) with an IrDA interface.

An IrDA UART for BASCOM-51

The IrDA interface looks rather complicated at first sight, especially considering the many layers of software. But at least at the lowest layer where bytes are physically communicated things are simple. The IrDA interface is a relatively normal serial interface using short pulses. Brief flashes of light are emitted for the start bit and for each zero data bit, i.e. in the bit positions where the RS232 TXD signal would be at a high level.

The pulse length is precisely specified. At data rates of up to 115,200 baud it must be at least 1.6 µs and at most 3/16th of the bit time. On the Palm the shortest possible pulse length of 1.6 µs is always used, which results in a power saving. Also, very short and intense pulses of infrared have a long range, as a specially-designed infrared receiver can easily distinguish them from ambient light, which generally contains no such short pulses.

Figure 1 shows serial signals on an RS232 interface compared with those on an IrDA interface. The IrDA receiver just needs to extend the light pulses suitably in order to create a signal that can be read by a normal RS232 interface.

An IR transmitter simply needs to convert the serial signals into brief pulses. Dedicated ICs are available for this purpose. In the Palm the function is already integrated into the processor, and the IrDA signals are generated using a special setting of the internal UART. This special mode is of course not available in the venerable 8051, and so a suitable

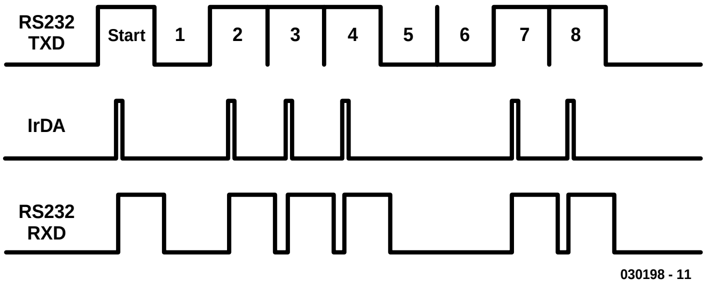


Figure 1. RS232 and IrDA signals.

IrDA UART must be implemented in software.

Simple IrDA routines can be implemented as assembler subroutines on the BASCOM-51. Subroutines Putir and Getir comprise the software UART. Data are passed between the assembler routines and the BASIC program using the accumulator by reading and writing the variable Acc. The transmit routine expects one byte in the accumulator. First it generates a pulse for the start bit on output P3.3. Here we use a pulse length of three instruction cycles, which, with an 11.059 MHz crystal, is about 3 µs. We avoid using the shortest possible pulse length in order to relax the requirements on the drive circuitry. After the start bit follows a delay of one bit time. Then, one after another, the eight bits in the accumulator are tested and for each zero bit a pulse is generated. Comments in **Listing 1** show the number of cycles taken by each instruction. The delay loop is set up so that the bit time is 104 µs which gives a data transfer rate of exactly 9600 baud.

The receive routine cannot poll the input signal directly as the short pulses could be missed. Instead, we use an interrupt. Input pin P3.2 is also the interrupt input Int0, and so we can write a suitable interrupt routine and enable interrupts on Int0. During initialisation we configure Int0 for edge-triggered interrupts so that each falling edge on the receiver input causes the interrupt service routine Puls to be called. The only action of the interrupt service routine is to set a flag bit. This bit remains set, indicating that an edge has been detected, until it is read and then reset. In this way we implement a kind of pulse stretching in software, and the very brief input pulses can be processed in a leisurely fashion.

The receive routine Getir is written in much the same way as would be done for a direct RS232 input. First the flag is cleared in case it has become set as a result of interference. The program then waits for a start bit. Once the flag is found to be set, the program waits for 150 % of a bit time, which takes us to the middle of the first data bit. Although a pulse (if any) will have occurred at the start of the bit time, testing for it in the middle of the bit time gives some tolerance to slight variations in data rates. If a pulse has not

Listing 1. Port access over IrDA.

```
‘_____________
‘ IrDA1.BAS
‘_____________

$regfile = “89s8252.dat”
Dim N As Byte
Dim Flag As Bit
Declare Sub Putirda
Declare Sub Getirda

Enable Interrupts
Enable Int0
Set Tcon.0       ‘INT0 falling edge.
On Int0 Puls Nosave ‘Nosave: timing!
While 1 = 1
  Getirda
  P1 = Acc
  Waitms 1
  Acc = P1
  Putirda
  Waitms 1
Wend
End

Sub Putirda
$asm
   clr   P3.3      ;1 start
   nop             ;1
   nop             ;1
   setb  P3.3      ;1
   mov   r2,#43    ;2
S1:
   djnz  r2,s1     ;43*2
   mov   r3,#8     ;2
S2:
   jb    ACC.0,S3  ;2,
              ;(10+86)*1.085µs=104µs
   clr   P3.3      ;1 data puls
   nop             ;1
   nop             ;1
   setb  P3.3      ;1
   sjmp  S4        ;2
S3:
   nop
   nop             ;delay
   nop
   nop
   nop
S4:
   mov   r2,#41    ;2
S5:
   djnz  r2,S5     ;43*2
   rr    a         ;1
   nop             ;1
   djnz  r3,S2     ;2
              ;(14+82)*1.085µs=104µs
   clr   Flag
   ret
$end Asm
End Sub


Sub Getirda
Flag = 0
$asm
Startbit:
   jnb   Flag,Startbit
   clr   Flag
   mov   r2,#60    ;1.5 bit times
S6:
   djnz  r2,S6     ;
   mov   a,#0
   mov   r3,#8
S7:
   rr    a         ;1
   jnb   Flag,S8   ;2
   sjmp  S9
S8:
   add   a,#128    ;1
   nop
   nop
   nop             ;delayh
   nop
   nop
   nop
S9:
   clr   Flag      ;1
   mov   r2,#40    ;2*40
S10:
   djnz  r2,S10    ;43*2
   djnz  r3,S7     ;2,
             ;(9+7+80)*1.085µs=104µs
   ret
$end Asm
End Sub

Puls:
 Flag = 1
Return
```

Listing 2. Transmitting and receiving port status using the Palm.

```
#irport.bas
open “com1:”,9600,ir as #5
draw -1
while 1
  input n
  put #5,n
  a= fn wait(0.1)
  i=fn serial(5)
  if i>1 then dummy=get$(#5,0)
  n =get$(#5,0)
  t$=str$(n)+”    “
  draw t$,75,60,2
  dummy =get$(#5,0)
wend
```

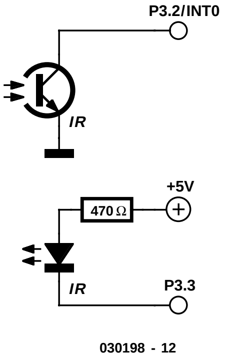


Figure 2. A simple IrDA interface for the 8051.

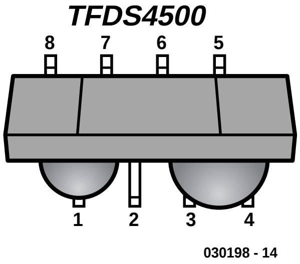


Figure 4. The surface-mount package of the TFDS4500.

been received, the appropriate bit is set in the accumulator; if not, the bit is cleared. The following seven bits are processed in the same way at regular intervals, the bits in the accumulator being shifted one place each time.

The main program outputs the received bytes directly on port 1. After the data have been output the port state is read back and transmitted on the IrDA port. A delay of one millisecond is inserted between transmission and reception in order to allow the receiver part of the interface settling time to recover from the high-power pulses of its own transmissions.

This example provides a complete application allowing writing to and reading from an I/O port. Possible uses include switching external loads and reading switch positions. If only inputs are being used, the value 255 should be sent in order to switch the port to input mode and immediately receive back the read values.

Initial construction of a simple IrDA device should not present any problems. Using the 89S8252 development board as a base, all that is needed in the simplest case is an IR diode at the transmit output (P3.3) and a phototransistor and the receive input (P3.2). The circuit in **Figure 2** shows this minimal implementation, which can be enhanced to give greater range by using a better IrDA front end.

Communication with the Palm

A programming language is needed to write quick and simple applications for small portable computers: BASIC is ideal. Fortunately, a BASIC interpreter for the Palm is available in the form of HotPaw BASIC, which can be experimented with for free. An evaluation version, unlocked for one month, can be downloaded from

www.hotpaw.com/rhn/hotpaw.

After that, you must either pay about $ 20 for the full version or continue with the evaluation version with one further small restriction: at most four programs can be kept in memory and run at one time.

The BASIC program examples shown here are available for download from the *Elektor Electronics* homepage in the form of plain text files. In order to use them, the listings must be copied into the Palm desktop MemoPad via the clipboard using a text editor. They will be transferred onto the device at the next HotSync.

All newer PDAs which run PalmOS are equipped with an IrDA interface, and it is not difficult to drive the interface using HotPaw BASIC. **Listing 2** shows an example for writing to and reading from the

Pin	Name	Function
1	IR cathode	–
2	RxD	Receiver data output, open collector
3	V_{CC1}	Power supply, 2.7 V to 5.5 V
4	GND	Ground
5	SC	Sensitivity control
6	NC	Not connected
7	TxD	Transmitter driver input, active high
8	IR anode	Anode connection for IR LED, connect to V_{CC} via 7 Ω to 14 Ω resistor

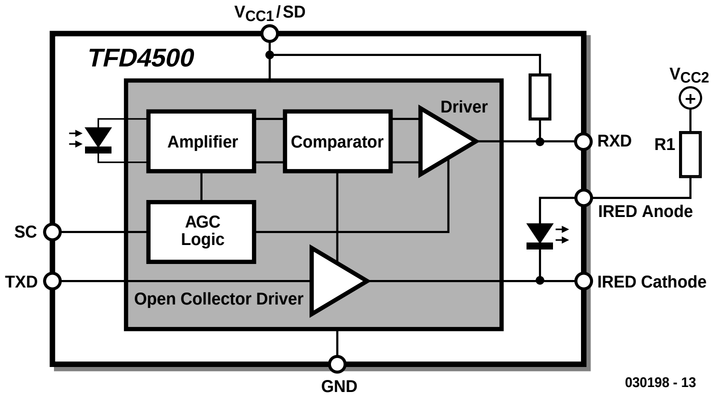


Figure 3. Block diagram of the TFDS4500.

IrDA receiver for the PC

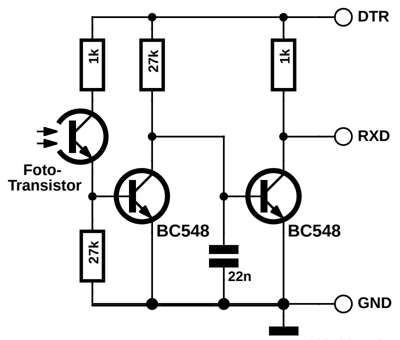


Figure A. A simple IrDA receiver for the PC.

Many readers will not have a Palm PDA or an IrDA interface on their PC, but would nevertheless like to experiment with wireless communication. In this year's December issue we'll describe a simple circuit using the Vishay TFDU6102. Simpler, but with just the receive channel, is the circuit in **Figure A**. This circuit is built entirely from standard components (a phototransistor, two NPN transistors, and a capacitor) and has little more to do than simply stretch the pulses. The circuit is designed for operation at 9600 baud and can be connected directly to the serial interface of a PC, from which it also derives its power supply. For this to work, the DTR signal must be turned on. Miracles should not be expected of this simple circuit: it is sensitive to ambient light, and so the light path should be shaded. The range is also limited to a modest 10 cm.

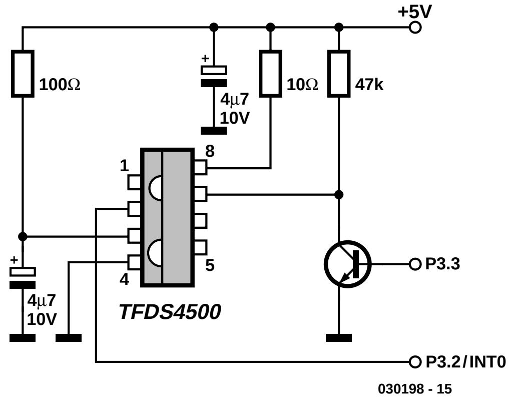


Figure 5. The improved IrDA adapter.

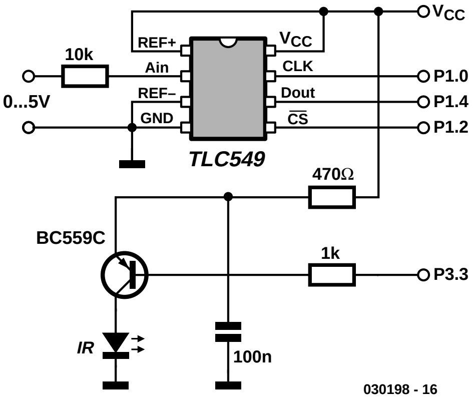


Figure 6. The analogue interface.

Listing 3. Transmitting analogue readings.

```
'____________-
'    IrDAtlc549.BAS
'____________-

$regfile = "89s8252.dat"
Dim N As Byte
Dim Addat As Byte
Declare Sub Adtlc549
Declare Sub Putirda


While 1 = 1
 Adtlc549
 Acc = Addat
 Putirda
 Waitms 250
Wend
End

Sub Adtlc549
 Clk Alias P1.0
 Cs Alias P1.2
 Adin Alias P1.4
 Dim Count As Byte
 Addat = 0
 Cs = 0
 Clk = 0
 Cs = 1
 Delay
 Cs = 0
 For Count = 1 To 8
  Shift Addat , Left , 1
  If Adin > 0 Then Addat = Addat + 1
  Clk = 1
  Delay
  Clk = 0
 Next Count
End Sub

Sub Putirda
$asm
   clr   P3.3
   nop
   nop
   setb  P3.3
   mov   r2,#43
S1:
   djnz  r2,s1
   mov   r3,#8
S2:
   jb    ACC.0,S3
   clr   P3.3
   nop
   nop
   setb  P3.3
   sjmp  S4
S3:
   nop
   nop
   nop
   nop
   nop
S4:
   mov   r2,#41
S5:
   djnz  r2,S5
   rr    a
   nop
   djnz  r3,S2
   ret
$end Asm
End Sub
```

ports on the Flash Microcontroller Board.

After each byte is output the program first reads back any echo byte that may have been received. To do this, the number of bytes in the interface buffer is first determined. If more than one byte has been received, the first byte in the buffer must be an echo of the byte that was transmitted, and the second is the reply from the microcontroller. After this a further dummy byte is read in order to clear any bytes from the buffer that may be the result of interference.

There are small hardware differences between the various Palm models. Although the Palm M100 will receive the echo of its own transmitted bytes, this is suppressed in the newer Palm Zire. The program shown here works with both models.

An integrated IrDA transceiver

The TFDS4500 integrated IrDA transceiver includes a complete IR receiver with photodiode, preamplifier, gain control and output stage. A transmitter LED with driver stage is also included. The package includes lenses to focus the light beam and is made of a filter material that is only transparent to infrared. This relatively inexpensive IC simplifies building an IrDA interface enormously. The receiver outputs the IrDA pulses in negative logic, whereas the transmitter must be driven with positive pulses. The IC is designed for surface mounting, but can relatively easily be mounted onto a prototyping board (**Figure 4**).

Table 1 shows the pinout of the IC. A capacitor should be connected between GND and VCC as close as possible to the device, since the sensitive receiver will otherwise be prone to interference. A resistor connected to pin 8 sets the current in the transmit LED. With a power supply voltage of 5 V a value of 10 Ω gives a peak current of around 250 mA. An additional 4.7 µF electrolytic capacitor is required in order to avoid interference being generated on the power supply.

The receiver output on pin 2 can be connected directly to the interrupt input of the microcontroller. In the transmit direction, however, an inverter is required, since the TFDS4500 is expecting positive pulses. Of course, the polarity of the pulses could be changed in the software, but this has the severe disadvantage that in the default state after reset all the processor's ports are set high. The IrDA transmitter will then be continuously active while the processor is initialising. It is better to insert a transistor to invert the signal.

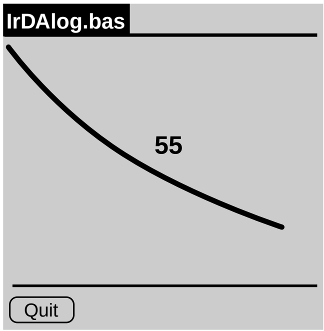


Figure 7. Data plot.

In comparison with the simple circuit of Figure 2 the TFDS4500 brings significant advantages. Reliable communication is possible at a range of up to around one metre.

Analogue-to-IrDA interface

Using the software UART described above opens up many possibilities. Here we will suggest a simple instrumentation application using a TLC549 analogue-to-digital converter. The microcontroller continuously transmits readings over the infrared port. Since only transmission, and not reception, is required, the interface can consist of just a simple infrared LED. A PNP transistor carries the high pulse currents, and a range of up to two metres can be achieved. Thanks to the short pulses used in IrDA, the average current consumption is only around 1 mA.

Listing 4. Displaying readings on the Palm.

```
#IrDAlog.bas
open “com1:”,9600, ir as #5
while 1
gosub AD
Ualt =U
draw -1
for x=1 to 158
gosub AD
draw x-1,150-Ualt/2,x,150-U/2
Ualt=U
t$=str$(U)+”    “
draw t$,75,60,2
next x
wend
close #5
end

sub AD
U=-1
while U=-1
  U =get$(#5,0)
wend
return
```

The program uses a delay of 250 ms and thus sends four readings per second, which is suitable for long-term measurements. If required the speed can be increased up to about 500 readings per second.

The following Palm program in HotPaw BASIC shows a typical application for this device. Readings are continuously plotted on the screen and can be observed as on an oscilloscope.

Timing in this example is entirely the responsibility of the program in the microcontroller, and is in this case set at four readings per second. Depending on the application, it would be possible to take more frequent readings which could then be selectively discarded in the program running on the Palm: this would allow the timebase to be configured on the PDA.

(030198-1)